Highlight & Summarize: RAG without the jailbreaks

Giovanni Cherubin
Microsoft Security Response Center

Andrew Paverd
Microsoft Security Response Center

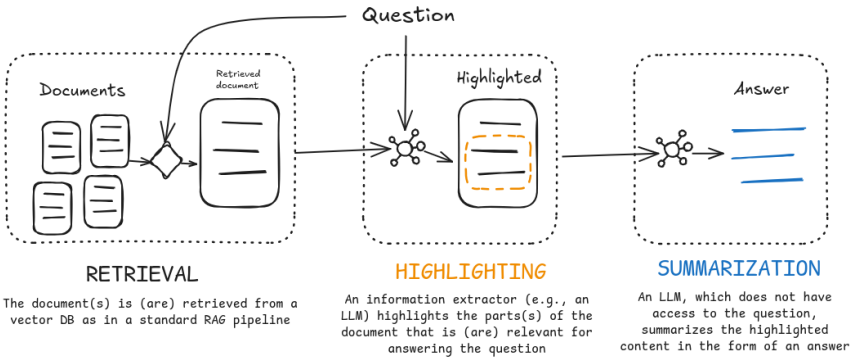

Figure 1: HS-enhanced RAG pipeline. A malicious user asking a question to this system cannot jailbreak the LLM, by design.

Abstract

Preventing jailbreaking and model hijacking of Large Language Models (LLMs) is an important yet challenging task. For example, when interacting with a chatbot, malicious users can input specially crafted prompts to cause the LLM to generate undesirable content or perform a completely different task from its intended purpose. Existing mitigations for such attacks typically rely on harden in the LLM's system prompt or using a content classifier trained to detect undesirable content or off-topic conversations. However, these probabilistic approaches are relatively easy to bypass due to the very large space of possible inputs and undesirable outputs.

In this paper, we present and evaluate *Highlight & Summarize* (H&S), a new design pattern for retrieval-augmented generation (RAG) systems that prevents these attacks *by design*. The core idea is to perform the same task as a standard RAG pipeline (i.e., to provide natural language answers to questions, based on relevant sources) without ever revealing the user's question to the generative LLM. This is achieved by splitting the pipeline into two components: a highlighter, which takes the user's question and extracts relevant passages ("highlights") from the retrieved documents, and a summarizer, which takes the highlighted passages and summarizes them into a cohesive answer. We describe several possible instantiations of H&S and evaluate their generated responses in terms of correctness, relevance, and response quality. Surprisingly, when using an LLM-based highlighter, the majority of H&S responses are judged to be *better* than those of a standard RAG pipeline.

1 Introduction

Retrieval-augmented Generation (RAG) is proving to be a sound and reliable solution for answering questions using documents from a knowledge base. From customer support systems to search engines, RAG combines the ability of LLMs to answer questions expressed in natural language with the efficiency of vector databases for retrieving information from large data repositories, to provide a robust production-ready solution for many applications. The core idea behind RAG is that, when a user asks a question, the system first retrieves a set of relevant documents from the knowledge base and passes these to a generative LLM together with the user's question; on this basis, the LLM produces an answer to the question, which is returned to the user.

There are, alas, many ways an adversary can exploit RAG systems, depending on which inputs the adversary can control. In this paper, we focus on the scenario where the knowledge base is trusted, while the users' inputs are untrusted and potentially malicious. A real-world example is a chatbot that is deployed by a company to answer questions based on the company's curated knowledge base (e.g., consisting of official FAQs or policy documents).

In this setting, a malicious user could attack the system by inputting specially crafted prompts to achieve various objectives. Firstly, they could try to *jailbreak* the system to make the LLM generate offensive content that harms the reputation of the company. A more subtle jailbreak could be to trick the LLM into generating unintended outputs that misrepresent the company's intent, for example, persuading the chatbot to offer discounts on the company's products. In some cases, the chatbot's output may constitute a legally binding statement from the company.[1] Alternatively, the malicious user could repurpose the generative LLM to perform some other task (e.g., asking questions unrelated to the company) – this is sometimes referred to as *model hijacking* or *wallet abuse*.

In this paper, we introduce and evaluate *Highlight & Summarize* (H&S), a new design pattern for RAG systems that prevents these attacks by design. Our approach consists of two components: First, the *highlighter* component inspects the retrieved documents and extracts passages from them that are relevant to answering the user's question – the digital equivalent of *highlighting* those passages. The highlighter can be instantiated using existing methods for extractive Q&A and passage retrieval, or using modern generative LLMs. Secondly, the *summarizer* component, which is an LLM, generates

[1] It was recently reported that Air Canada had to pay compensation to a customer for misleading information provided by their FAQ chatbot. https://www.theguardian.com/world/2024/feb/16/air-canada-chatbot-lawsuit.

a coherent answer by summarizing the highlighted passages. **Crucially, the summarizer never sees the user's question**. This mitigates the above attacks whilst maintaining the utility of the RAG system on its intended question-answering task.

The remainder of this paper is organized as follows. We first present background and discuss related work (section 2) and then define our precise threat model (section 3). We then describe different implementations of the H&S design (section 4). Next we explain our experimental setup (section 5), evaluate the individual components of H&S (section 7), and evaluate the full H&S system in comparison to a current RAG pipeline (section 6). Finally, we evaluate the security of H&S (section 8), and discuss future work and open questions (section 9).

We release both the code and the data we used, including the responses generated by the various RAG pipelines and the associated ratings given by the LLM-as-Judges. Our aim in doing so is to enable further analysis of H&S, without incurring the monetary and environmental costs of rerunning our experiments.
https://github.com/microsoft/highlight-summarize

2 Background and related work

Retrieval-augmented generation (RAG). At a high level, the intended interaction between as user and a typical LLM-based RAG application would be:

(1) The user submits a question via a chat-style interface.
(2) The application searches its knowledge base and retrieves documents related to the user's question.
(3) The retrieved documents are concatenated to the user's question and the combined text is input into an LLM.
(4) The LLM generates a response, which the application returns to the user.

This type of RAG application is widely used, for example in customer support services.[2,3] The benefits of using RAG in such scenarios include the ability to ground the LLM's response on company-specific information, reduce hallucinations, and update the knowledge base without retraining the LLM. However, as explained above, such systems are still vulnerable to jailbreaking and model hijacking by malicious users.

Preventing jailbreaks and model hijacking. Several techniques have been proposed to detect or prevent jailbreaking of LLM-based systems, including those used in RAG. The main classes of defenses include: finetuning the LLM to reduce the risk of undesirable output; using defensive system prompts to increase the difficulty of jailbreaking [19, 21]; applying different types of classifiers to the inputs to detect potential jailbreaks [4, 12, 13, 25, 28]; and preprocessing the input to remove or reduce the impact of potential jailbreaks [13, 17, 31, 32]. To mitigate model hijacking (i.e., repurposing the LLM for a different task), techniques such as NeMo [31] support programmable guardrails that allow the application owner to specify dialogue flows by canonicalizing inputs. Our proposed solution, H&S, takes a completely different approach by ensuring that the user's input is never input to the summarizer LLM.

[2] https://careersatdoordash.com/blog/large-language-modules-based-dasher-support-automation/.
[3] https://medium.com/tr-labs-ml-engineering-blog/better-customer-support-using-retrieval-augmented-generation-rag-at-thomson-reuters-4d140a6044c3.

System-level defenses. Similarly to defenses such as CaML [10] and FIDES [9], H&S is a system-level defense in that it deterministically limits the attacker's capabilities. However, H&S focuses on the opposite threat model, where the retrieved data is trusted but the user inputs are not. H&S is a realization of the Context-Minimization design pattern [5].

Passage retrieval and extractive Q&A. Passage retrieval and extractive Q&A are well-established research fields in the domain of natural language processing (NLP). Passage retrieval is the process of extracting from a set of documents one or more texts that are relevant to answering a question. Several techniques have been developed, with applications in the medical and legal sectors [6, 15, 22, 23]. With a similar goal, extractive Q&A aims to select (brief) portions of text from a larger document in order to answer a question. Extractive Q&A methods are largely based on modern NLP architectures, with BERT-like models being the most successful [18, 26, 35]. Passage retrieval and extractive Q&A are typically combined in practical systems [16, 29]. In an H&S pipeline, the highlighter has a very similar goal to passage retrieval and extractive Q&A. Indeed two of the instantiations of the highlighter we describe in section 4 use extractive Q&A models.

3 Threat model and assumptions

Some users of a RAG application may be adversarial. We assume a strong adversary who has full knowledge of all the documents in the knowledge base; for example, this may be the case if the knowledge base consists of published documentation or FAQ web pages. The adversary is able to submit arbitrary questions and observe the responses. The adversary may want to achieve the following goals:

- Jailbreak the application and cause it to generate undesirable output. This could include content that causes reputational damage to the application owner, or even constitutes an unintentional yet legally enforceable commitment from the application owner.
- Repurpose the application to perform a different task. For example, using a customer service chatbot to summarize large amounts of text, at the expense of the application owner.

The goal of the application owner is to prevent both of the above classes of attacks. We assume the application owner has inspected all documents in the knowledge base and confirmed that they contain only trustworthy information. This is a realistic assumption as the application owner has full control of the knowledge base and can apply arbitrary preprocessing and filtering. Note that we do not aim to defend against other types of attacks on RAG systems, such as extracting information [14, 27] or poisoning the knowledge base [36, 38].

4 Highlight & Summarize

We propose *Highlight & Summarize* (H&S), a new design pattern for RAG systems to mitigate jailbreaking and model hijacking attacks. The main design principle is that a (malicious) user must be unable to provide direct inputs to the LLM that generates the response. We achieve this by separating the generative Q&A process into two steps, as illustrated in Figure 1:

- **Highlighting:** The highlighter component takes the retrieved documents and selects ("highlights") text passages from these documents that are relevant to answering the user's question.
- **Summarization:** The summarizer component takes only the text selected by the highlighter and summarizes it into a coherent answer to *a* question, which is returned to the user. The user's question is never shown to the summarizer.

By virtue of this design, malicious users are unable to directly influence the outputs of the system – which are provided by the summarizer. In section 8, we further evaluate the security of this design pattern, as well as discuss possible attacks and mitigations.

4.1 H&S implementations

H&S can be implemented in multiple ways. In this section we describe our implementation of the summarizer component and three different implementations of the highlighter. We refer to the combination of any of these highlighters with the summarizer as an *H&S pipeline*.

Summarizer (common across all H&S implementations.) The summarizer is a zero-shot prompt-tuned LLM that is tasked with i) guessing what question the highlighted text was intended to answer, and ii) reformulating the highlighted text in form of an answer. Only the answer is returned to the user. The guessed question is not returned to the user – it's purpose is to ground the generated answer, and to aid in evaluation. We use Azure OpenAI's structured output option[4], which forces the LLM to give a response matching a desired format (in this case, `{"guessed_question": str, "answer": str}`); we found this helps with the quality of the responses, and allows us to evaluate what question the LLM guessed. Unless otherwise specified, we employ OpenAI's `GPT-4.1 mini` [2] as our default generative LLM.

H&S Baseline. This highlighter is a zero-shot prompt-tuned LLM that is tasked with extracting relevant information from the retrieved documents. Since the output of the highlighter LLM might deviate slightly from the exact text in the retrieved documents (e.g., due to the model's internal randomness), we use fuzzy string matching between the LLM's output and the documents to identify the precise text from the documents to be highlighted We use `RapidFuzz`[5] with a threshold of 95 in our experiments.

H&S Structured. This highlighter improves upon H&S Baseline by asking the highlighter LLM to first output an answer to the user's question and then to highlight the relevant text from the retrieved documents. We again use Azure OpenAI's structured output to enforce the following format:
`{"answer": str, "text_extracts": list[str]}`.
The generated answer is not passed to the summarizer. We observed that asking the highlighter LLM to first produce this output helped with grounding its responses, thereby producing better highlights from the text. As with H&S Baseline, we again use fuzzy string matching to ensure that the highlighted text is taken directly from the retrieved documents.

[4] https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/structured-outputs.
[5] https://github.com/rapidfuzz/RapidFuzz

H&S DeBERTaV3. This highlighter is an extractive Q&A model from the BERT family. In our experiments, we use DeBERTaV3, which uses disentangled attention and a better mask decoder to improve upon BERT and RoBERTa. We use two versions of this model: *H&S DeBERTaV3 (SQuAD2)* is a DeBERTaV3 model that was fine-tuned[6] on the SQuAD2 dataset [30]. Since this model was fine-tuned for short-span extractive Q&A, as encouraged by the SQuAD2 dataset, its performance in our experiments was lacking: the highlighter of an H&S pipeline is best served by a tool that outputs longer span passages to a question. For this reason, we also employ *H&S DeBERTaV3 (*`RepliQA`*)*, which is a DeBERTaV3 model that we fine-tuned on the `RepliQA` dataset (see section 5). We train the model to return the gold passage of an answer, rather than the expected answer, which significantly improves the performance of the H&S pipeline. Fine-tuning this model on splits 0-2 of the `RepliQA` dataset (section 5) took around 7 hours on an NVIDIA A100 GPU.

5 Experimental setup

We describe the datasets and metrics used in our experiments.

5.1 Datasets

Evaluating generative LLMs for RAG pipelines requires extreme care: while we may want to judge the LLM's ability at responding based on source documents, it often happens that the LLM answers based on its training data instead. For this reason, we conduct our main experiments using the `RepliQA` dataset [20]. This is a human-created dataset that contains questions based on natural-looking documents about *fictitious* events or people. By doing so, we ensure that the performance is not affected by the ability of LLMs to memorize their training data.

Importantly, each entry in the `RepliQA` dataset contains a *gold passage*, called `long_answer`, which is a substring of the retrieved document, selected by a human annotator, that is relevant to answering the question. We use this field as part of our performance measurements, as well as for fine-tuning (on a separate split of the dataset) a DeBERTaV3 extraction Q&A model to implement the H&S DeBERTaV3 (`RepliQA`) pipeline.

The `RepliQA` dataset consists of 5 splits (numbered from 0 to 4), which were released gradually over a year. Each split contains 17,955 examples. In our evaluation, we used the most recent split (`split_3`, which was released on April 14th, 2025). During data analysis, we observed 10 mislabeled instances in this split, which we corrected manually.

We also include experiments based on the `rag-mini-bioasq` dataset, henceforth `BioASQ`, which is a subset of the training dataset that was used for the BioASQ Challenge.[7] This dataset contains biomedical questions, as well as answers and documents. Since we do not use this dataset for training, we use both the training and test splits for evaluation, which gives a total of 4,719 samples.

5.2 Evaluation metrics

We primarily compare each H&S pipeline's answer with the expected answer from the dataset using several metrics. For some

[6] https://huggingface.co/deepset/deberta-v3-base-squad2.
[7] https://www.bioasq.org.

Table 1: Metrics used for evaluating individual H&S components and the full H&S pipeline.

Name	Type	Description	Ref
Recall	Token	Proportion of tokens in the reference answer are present in the model's response.	[3]
K-Precision	Token	Proportion of tokens in the model's reponse that are present in the gold passage.	[3]
Poll Multihop Correctness	LLM	Correctness of a generated response against a reference answer using few-shot learning.	[34]
Reliable CI Relevance	LLM	Relevance of a passage to a query based on a four-point scale: Irrelevant, Related, Highly relevant, Perfectly relevant.	[24]
MTBench Chat Bot Response Quality	LLM	Quality of the response based on helpfulness, relevance, accuracy, depth, creativity, and level of detail, assigning a numerical grade.	[37]
ComparisonJudge	LLM	Compare two answers to the same question and select either a winner, a tie, or a tie where neither answer is acceptable.	Ours

experiments on the `RepliQA` dataset, we also compare the outputs of the highlighter against the gold passage (`long_answer`). We evaluate the quality of the responses based on three types of evaluation metrics, as summarized in Table 1.

Token-based metrics. We use quantitative comparisons between the tokens of the expected answer (or gold passage) and the provided answer. Specifically, based on the work by Adlakha et al. [3], we use token-based metrics to evaluate the correctness ("Recall") and faithfulness ("K-Precision") of an answer.

LLM-as-a-Judge. We use prompt-tuned LLMs that rate the responses with respect to various criteria [7, 37]. Specifically, we adopt three standard LLM-as-a-Judge implementations, so as to provide a diverse judgment [24, 34, 37].

Comparison judge. We use a zero-shot prompt-tuned LLM with the task of deciding a "winner" between two alternative answers to a question. This judge also has the option to declare a "tie" between the answers, or to decide that neither answer is acceptable. In our implementation, we randomize the order of the two answers to mitigate any potential ordering bias.

6 Evaluating the full H&S pipeline

We first evaluate the full H&S pipelines in comparison with a standard ("vanilla") RAG system. Since H&S only modifies RAG generation step, we hold the retrieval step constant for each comparison; that is, we compare the pipelines on their ability to answer questions based on the same set of retrieved documents. We compare the pipelines using all three LLM-as-Judges and the ComparisonJudge, as well as on their ability to decline to answer when they believe no answer can be provided. We also compare time measurements for each of the pipelines (Table 4). Although the absolute time measurements are specific to our experimental setup, this shows the relative time overhead of each pipeline.

6.1 Direct pairwise comparisons

We use the ComparisonJudge to perform direct pairwise comparisons between all pipelines, including standard RAG. Table 2 shows percentage of wins for each pipeline (excluding ties), as well as the pipeline's Elo score. The Elo score, invented in the context of chess for ranking players based on 1-to-1 matches [11], has recently become a popular metric for comparing LLMs (e.g., by Chiang et al. [8]). The full pairwise results are shown in Figure 2 and Table 9.

Table 2: Direct pairwise comparison of all pipelines by the ComparisonJudge LLM. Ties are omitted.

		Wins	Elo Score
`RepliQA`	H&S Structured	59%	1211
	H&S Baseline	54%	1147
	H&S DeBERTaV3 (`RepliQA`)	32%	992
	Vanilla RAG	32%	958
	H&S DeBERTaV3 (SQuAD2)	8%	688
`BioASQ`	H&S Structured	69%	1257
	H&S Baseline	62%	1109
	H&S DeBERTaV3 (`RepliQA`)	24%	945
	Vanilla RAG	42%	899
	H&S DeBERTaV3 (SQuAD2)	10%	787

We observe that the LLM-based H&S pipelines (Structured and Baseline) vastly outperform the other pipelines. **This is a surprising result because it suggests that H&S simultaneously gives security against malicious users as well as *better* response accuracy than standard RAG.**

6.2 Correctness, relevance, and quality

We compare the pipelines on the basis of three LLM-as-Judges that independently evaluate correctness, relevance, and quality (as detailed in Table 1).

First, as shown in Figure 3, we count the number of times each pipeline received the highest rating out of all the pipelines (including tied highest). We refer to this as a "win" for that pipeline. We observe that, whereas the pipelines are very similar in terms of relevance and quality of the responses, there are notable differences in correctness. On `RepliQA`, H&S Structured obtains a better result (13,698 wins out of 17,955 examples), followed by H&S Baseline (13,401), and Vanilla RAG (13,332). We observe a similar behavior on `BioASQ`. Interestingly, on the `BioASQ` dataset we note that H&S DeBERTaV3 (SQuAD2) outperforms H&S DeBERTaV3 (`RepliQA`);

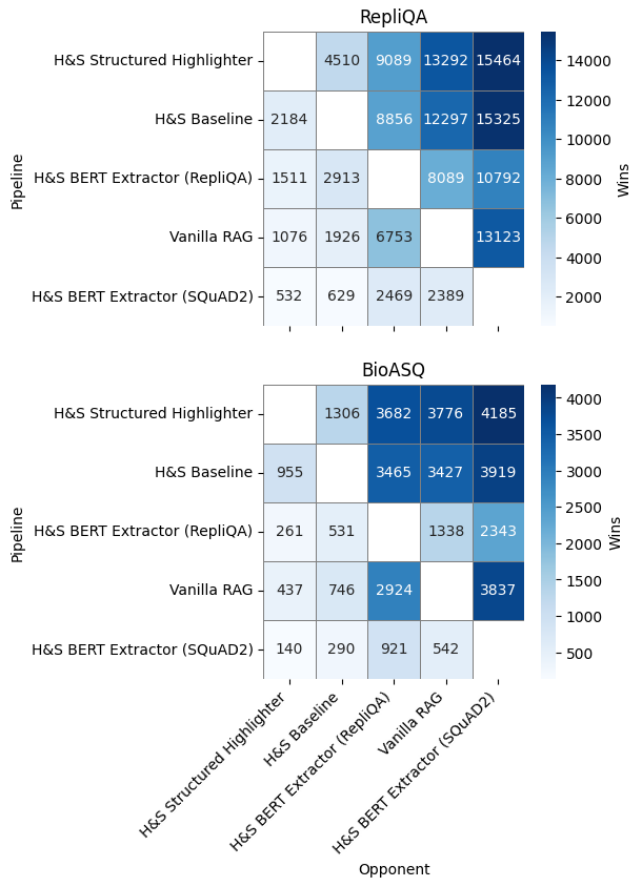

Figure 2: Wins of each pipeline in the direct pairwise comparison by the ComparisonJudge. Ties are omitted.

this is likely due to the fact that the latter was fine-tuned on (a separate split of) `RepliQA`. Figure 4 shows the distribution of ratings from each judge. We discuss each judge separately.

Correctness. Poll Multihop Correctness evaluates the correctness of a generated answer against the reference answer. Most pipelines output >80% correct answers. We observe that H&S DeBERTaV3 (SQuAD2) performs poorly on the `RepliQA` dataset, likely due to its base highlighter being fine-tuned for returning short answers. On the other hand, we observe that the same base model fine-tuned on a separate split of the `RepliQA` dataset performs rather well on the `RepliQA` dataset, suggesting that the performance of such extractive Q&A models can be further improved through fine-tuning. However, the same pipeline performs poorly on the `BioASQ` dataset. This suggests that the data distribution highly impacts performance, and that more varied fine-tuning datasets should be used in practice.

Relevance. The Reliable CI Relevance explicitly captures how relevant an answer is to a question. We observe minimal discrepancy as all pipelines receive a nearly perfect relevance on average (2.72 out of 3). We observe slightly better performance by the vanilla RAG pipeline, followed by the H&S DeBERTaV3 (SQuAD2).

Table 3: Precision and Recall for declined answers.

Pipeline	Precision	Recall	F1
H&S DeBERTaV3 (`RepliQA`)	0.85	0.99	0.91
H&S Structured	0.94	0.39	0.55
H&S DeBERTaV3 (SQuAD2)	0.55	0.48	0.51
Vanilla RAG	0.95	0.32	0.48
H&S Baseline	0.58	0.20	0.29

Response quality. MTBench Chat Bot Response Quality evaluates the responses on a scale from 1 to 10. We observe its rating for all pipelines is just below (resp. above) 5 out of 10 for the `RepliQA` (resp. `BioASQ`) datasets. Based on manual inspection of the judge's explanations, we observe that the judge tends to demand additional information beyond what is available in the retrieved document. This suggests the judge is better suited for rating LLMs on open ended questions without source documents, rather than in RAG settings.

6.3 Decline to answer

Around 10% of questions in the `RepliQA` dataset cannot be answered based on the provided document. We evaluate how well the various pipelines declined to answer these questions.

In Table 3, we report K-Precision and Recall for each pipeline in terms of declining to answer. We observe that DeBERTaV3-based models fare well, which is possibly due to the fact that their fine-tuning considers the no-answer case explicitly. The H&S DeBERTaV3 (`RepliQA`) performs particularly well; this may be partially due to the fact that it was trained on data with a similar distribution as the test set. H&S Structured also does relatively well, possibly because the structured output option helps its reflection process.

Nevertheless, the absolute performance of all the pipelines suggests there is still significant potential for improvement in this aspect. Future H&S implementations should consider either better fine-tuning or few-shot prompt-tuning to improve this metric.

6.4 Processing time

Finally, we measure the time taken by each pipeline to generate an answer, averaged over 40 questions, as shown in Table 4. The absolute time values are specific to our setup, but the relative differences show that, as expected, the LLM-based H&S pipelines require more time to generate an answer. We note that our implementation has not yet been optimized for performance.

Table 4: Processing time for one question, averaged over 40 examples.

Pipeline	**Time (s)**
Vanilla RAG	0.49
H&S DeBERTaV3 (`RepliQA`)	0.75
H&S DeBERTaV3 (SQuAD2)	0.76
H&S Baseline	2.26
H&S Structured	3.05

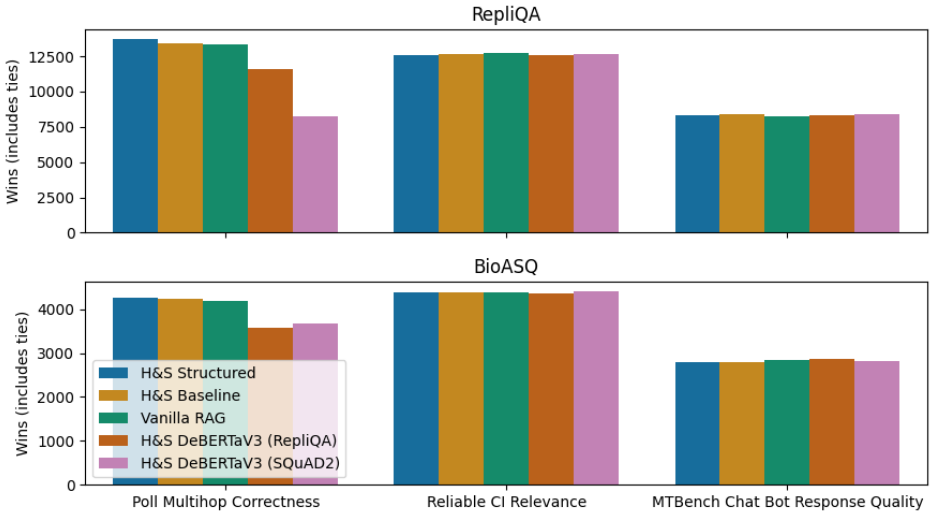

Figure 3: Number of times each pipeline had the highest rating, including ties, out of all pipelines, according to LLM-as-Judges.

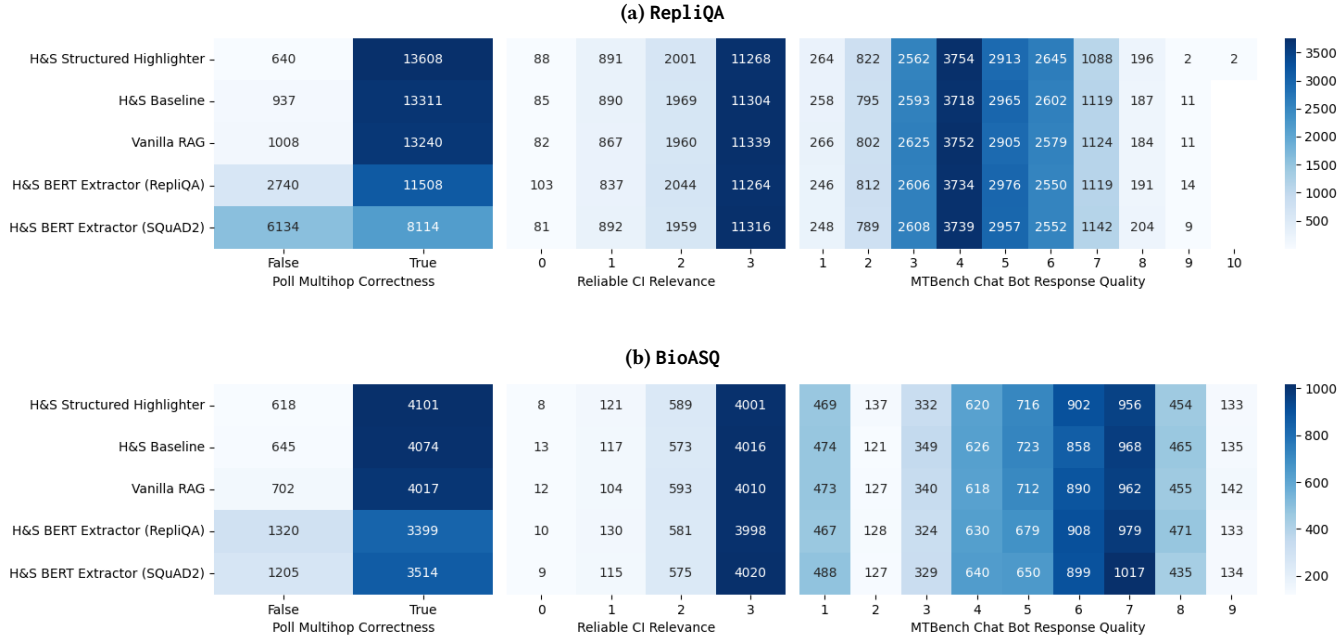

Figure 4: Response's evaluation via LLM-as-Judges on the two datasets, measuring: correctness, relevance, and quality.

7 Evaluating H&S components

In this section we evaluate the individual components of an H&S pipeline, to provide deeper insights into *why* this approach can achieve the results presented above. First, we explore whether H&S is needed at all, or whether a simple passage retrieval or extraction Q&A pipeline (i.e., highlighter without the summarizer) may suffice. Next, we compare our different highlighter implementations in terms of their K-Precision and Recall. Finally, we investigate whether the summarizer can recreate the original question based solely on the outputs of a highlighter.

7.1 Do we need a generative summarizer?

First, one may wonder: do we actually need the summarizer in the pipeline? Given the long history of passage retrieval and extractive Q&A, a simpler highlighter, which can be implemented on the basis of an extractive Q&A model, may perform well in Q&A tasks.

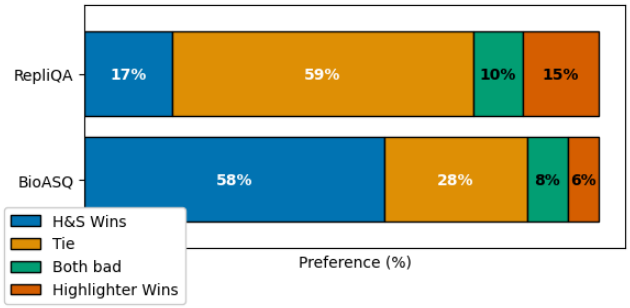

Figure 5: Do we need H&S, or can we just do highlighting (e.g., extractive Q&A)? The preference is based on an LLM-based ComparisonJudge.

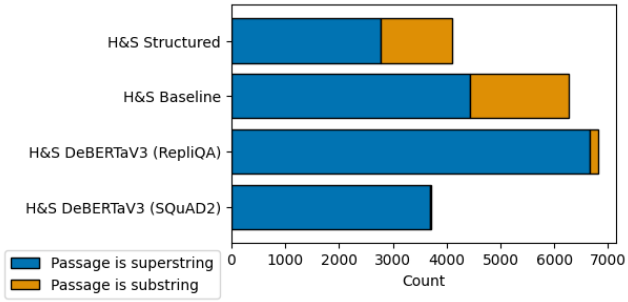

Figure 6: Comparison of the highlighter's output text to the human-curated gold passage in the `RepliQA` dataset. "Passage is a sub/superstring" means that the gold passage is a sub/superstring of the highlighted text.

To answer this question, we compare the outputs of the individual highlighters against the response given by the full H&S pipeline using a ComparisonJudge. As shown in Figure 5, our results indicate that using a generative LLM (summarizer) after the highlighting step can lead to noticeable improvements. This is particularly evident on the `BioASQ` dataset, where the H&S outputs were preferred in 58% of cases versus only 6% for the highlighter alone. Based on a qualitative manual inspection of the ComparisonJudge's explanations accompanying its ratings, we observe that the H&S pipeline is typically preferred for its more "natural" and "well-structured" answers, whereas the highlighter is preferred for more "concise" and "direct quote" answers.

7.2 How good are the highlighters?

We now compare the various highlighter implementations in terms of their K-Precision and Recall with respect to the gold passage. Since this analysis requires a gold passage, we are only able to use the `RepliQA` dataset.

Table 5: Comparison between highlighter implementations with respect to the gold passage (`RepliQA` dataset.)

Implementation	K-Precision	Recall
H&S Structured	0.84	0.76
7 H&S Baseline	0.84	0.65
H&S DeBERTaV3 (`RepliQA`)	0.80	0.36
H&S DeBERTaV3 (SQuAD2)	0.55	0.22

As shown in Table 5, we observe that LLM-based highlighters significantly outperform those based on DeBERTaV3, especially in terms of Recall. However, as discussed in section 6, LLM-based highlighters incur higher computational overheads, which may be a consideration in practical applications. We did not focus on optimizing the performance of the DeBERTaV3 models, and we hypothesize that there may still be scope for further improvement. An encouraging result in this direction is that fine-tuning DeBERTaV3 for the specific task of long-context highlighting (`RepliQA` dataset) improves performance compared to the same model fine-tuned on SQuAD2 (short answers). Results in Table 5 also indicate that using structured outputs for LLM-based highlighters improves performance (Recall). However, as reported in section 6, this incurs additional computational costs.

We further evaluate highlighters on a stricter metric: we count how many times the highlighter's output is either a substring or a superstring of the gold passage (although one should bear in mind that the span of the human-chosen gold passage is somewhat arbitrary). Figure 6 shows that, in this case, the DeBERTaV3 highlighter that was fine-tuned on `RepliQA` has the best performance. We attribute this to the fact that, even if its fine-tuning was done on a separate split of the `RepliQA` dataset, the model may have captured the style of gold passage selection of the human annotators for `RepliQA`.

7.3 Can the summarizer guess the question?

To gain a deeper understanding of the internal workings of an H&S pipeline, we investigate whether the summarizer can guess the user's original question, based solely on the text provided by the highlighter. We start by noting that this problem is, by its nature, ill-posed. For example, consider the passage *"In his later years, Kant lived a strictly ordered life. It was said that neighbors would set their clocks by his daily walks"*.[8] This text could be a plausible answer to several realistic questions, including *"Was Immanuel Kant a creature of habit?"*, *"What philosopher is best known for their punctuality?"*, or *"Did Kant enjoy walking?"*.

When implementing the summarizer (section 4), we ask the LLM to output `guessed_question`, a field that is not used by the H&S pipeline, but which helps with grounding the model. Table 6 shows the K-Precision and Recall of `guessed_question` with respect to the true question. These experiments are carried out for H&S Structured pipeline. As expected, the summarizer is rarely able to guess the question correctly. However, its average performance is far from zero, with better pipelines achieving higher scores.

In Table 7, we report some of the *worst* examples, which highlight the ill-posedness of the problem. In section 9 we consider whether monitoring the guessed question can help with controlling the performance of H&S pipelines.

[8] https://en.wikipedia.org/wiki/Immanuel_Kant.

Table 6: Can the summarizer guess the user's question?

	Pipeline	K-Precision	Recall
RepliQA	H&S Structured	0.55	0.49
	H&S Baseline	0.46	0.39
	H&S DeBERTaV3 (RepliQA)	0.41	0.34
	H&S DeBERTaV3 (SQuAD2)	0.29	0.21
BioASQ	H&S Structured	0.38	0.60
	H&S Baseline	0.34	0.53
	H&S DeBERTaV3 (RepliQA)	0.34	0.46
	H&S DeBERTaV3 (SQuAD2)	0.24	0.33

Table 7: Worst question guesses by the summarizer of the H&S Structured pipeline. K-Precision for all was 0.

Real question	Guessed question
What is the main argument of the opinion piece titled 'The Critical Shield'?	Why should mandatory automatic software updates be implemented on all consumer devices?
Which local candidate became notorious for tweeting vague statements like 'New policies on the horizon #blessed #nofilter'?	Who is Harvey Peterson and why did he gain notoriety?
What was the primary topic of discussion by Mark Downey at the Online Marketing Summit in San Diego on October 6, 2023?	How can social media benefit small and medium-sized enterprises (SMEs)?

8 Security considerations

Fundamentally, all forms of jailbreaking and model hijacking attacks involve some type of adversarial input to an LLM, which potentially causes the LLM to generate undesirable outputs or perform an unintended task. In the widely-used setting of a RAG-powered system with a trusted knowledge base, the key idea of H&S is to ensure that the adversary cannot provide *direct* inputs to the generative LLM that produces the final output (i.e., the summarizer). In subsection 8.1 we demonstrate how this design mitigates all jailbreaks from an existing dataset.

It is important to note that even H&S cannot *guarantee* the absence of jailbreaks or model hijacking because the adversary still has some degree of *indirect* influence over the inputs to the generative LLM through the highlighted text. Depending on the contents of the knowledge base, it may be possible for the adversary to cause the highlighter to select text that triggers undesirable behavior in the generative LLM. We describe this *adversarial highlighting* adaptive attack in subsection 8.2. Another potential adaptive attack could involve the adversary influencing the highlighter to select only part of the text necessary to produce the correct answer. This could lead to the generated answer being correct but incomplete. We describe this *incomplete highlighting* adaptive attack in subsection 8.3.

Table 8: Evaluation on the LLMail-Inject dataset (Scenario 2), showing the percentage of all jailbreaks that succeeded in calling the prohibited tool and the percentage of all jailbreaks that succeeded in calling the tool with valid arguments.

	Tool called	Arguments valid
RAG	81%	53%
H&S (Highlighter only)	93%	63%
H&S (Full)	0%	0%

8.1 Non-adaptive attacks

Evaluating LLM jailbreaks is notoriously challenging because it requires an automated method for detecting whether the jailbreak succeeded. Inspired by the recent LLMail-Inject challenge [1], we use *tool calling* as a well-defined attack target. We give the LLM the ability to call a (simulated) email sending tool by outputting a specific string (e.g., `send_email()`). For example, a customer service chatbot may have this type of tool to send an email to the customer support team if a user asks about certain topics, but the users should not be able to control when the LLM triggers this tool call. The attacker's goal is therefore to cause the LLM to trigger this tool call[9]. For purposes of evaluation, we give *both* LLMs (i.e., highlighter and summarizer) the ability to trigger this tool call. However, in practice only the summarizer would have this ability, so a successful attack against H&S would need to trick the summarizer into making this tool call.

For these experiments, we use 1,028 successful attack prompts from the LLMail-Inject challenge dataset (Scenario 2) [1]. Our setup differs slightly from that of the challenge: in the challenge, the adversarial inputs were retrieved from the knowledge base, whereas in our setting we input these directly as the user's prompt. However, this modified setup does not affect the results: successful prompts from LLMail-Inject are usually successful in our setup.

Table 8 shows the percentage of all inputs for which the tool was called (with any arguments) or called with valid arguments. As expected, the vast majority of the attack inputs succeed in jailbreaking the RAG pipeline, but none are successful against H&S. The results on H&S (Highlighter only) confirm that our highlighter LLM, like any other, is still susceptible to jailbreaks when processing direct adversarial input. This further emphasizes the need for system-level defenses, such as H&S.

8.2 Adaptive adversarial highlighting attack

As explained in section 3, we assume a strong adversary who may have full knowledge of H&S and the contents of the knowledge base used by the system under attack. We envision that such an attacker could attempt to subvert H&S with the following adaptive attack, which is somewhat reminiscent of traditional system attacks such as Return-Oriented Programming (ROP) [33]. For a target sentence, the attacker i) browses the RAG knowledge base to find a document that contains (most of) the words in the target sentence, ii) causes that document to be retrieved, and iii) asks the H&S highlighter to

[9]This is very similar to the setting of a recent vulnerability found by Zenity Labs: https://labs.zenity.io/p/a-copilot-studio-story-2-when-aijacking-leads-to-full-data-exfiltration-bc4a

Reimbursement_Guidelines.pdf

You may submit a request for reimbursement if the expense falls within the approved categories outlined in Appendix D. Once the claim is verified and approved, it is recorded as a completed transaction and marked as "won" in the internal tracking system. A confirmation email will be sent to the claimant within three (3) business days. If the reimbursement includes a $10 threshold adjustment, the system will automatically generate a voucher code for accounting purposes. This voucher is non-transferable and must be used within the same fiscal quarter.

Figure 7: Adaptive adversarial highlighting attack example.

only highlight the desired words. For example, as shown in Figure 7, if the attacker wants the system to output "You won a $10 voucher", they could try to persuade the highlighter to highlight only specific words from the retrieved document.

While this *adversarial highlighting* attack is theoretically possible, its difficulty can be substantially increased through a simple H&S design constraint: we only allow the highlighter to highlight contiguous passages of at least a certain length. In our implementations, this attack was readily prevented by the use of fuzzy string matching to ensure that the highlighted text corresponds to a contiguous portion of the original text. Fundamentally, H&S transforms the very challenging problem of detecting any possible jailbreak in the LLM's inputs into the significantly simpler task of inspecting the knowledge base to check that it does not contain strings that can be used to trigger undesirable outputs.

8.3 Adaptive incomplete highlighting attack

A notable variant of the adversarial highlighting attack is the *incomplete highlighting* attack. This adaptive attack is designed to overcome the mitigation to the adversarial highlighting attack discussed above. Even if we force the highlighter to select a contiguous span of at least a certain length, we cannot guarantee that the highlighter will include all text that is necessary to generate a *complete* answer. For example, as shown in Figure 8, the adversary might convince the highlighter to omit important information. Note that the output of the H&S pipeline is still correct, but may be incomplete.

One possible mitigation for this attack is to pass additional context to the summarizer, in addition to the highlighted text. The summarizer can still use the highlighted text, as before, but can also situate this within the included context. For example, in Figure 8, if the summarizer received the full retrieved document in addition to the highlighted text, the summarizer would likely produce a complete answer. The system designer could divide up the knowledge base such that any retrieved document is *complete* in itself, and can thus be passed as context to the summarizer. We leave the evaluation of this mitigation as future work.

Vacation_Guidelines.pdf

You may submit a vacation request if:

- You have not exceeded your quota for the year;
- You have your manager's approval.

Your request will not be approved unless both the above conditions are satisfied.

Figure 8: Adaptive incomplete highlighting attack example.

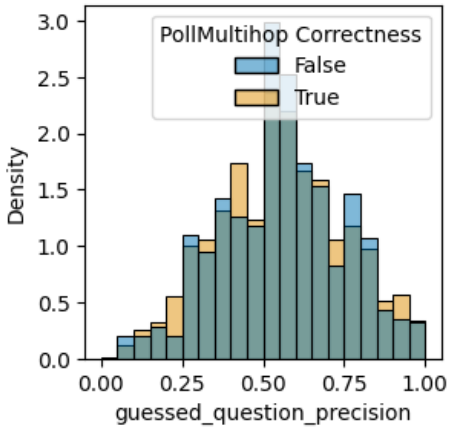

Figure 9: Lack of correlation between the ability of the summarizer to guess the question and the performance of the H&S Structured pipeline. Pearson correlation: -0.004.

9 Discussion and future directions

This study of a new design pattern opens up a large number of research questions and directions to explore. We discuss some of the challenges that we leave open for future work.

Monitoring the guessed question. Future implementations of H&S pipelines could monitor the similarity between the user's question and the question guessed by the summarizer. If there is too much divergence, the system could be programmed to refuse to answer, or to adapt the summarizer's task accordingly. This would not impact the security of the pipeline since the summarizer still does not see the user's question directly. In our current implementation, we found no significant correlation between the pipeline's performance and the summarizer's ability to predict the question. For example, Figure 9 shows the relationship between the full pipeline's performance (measured via an LLM as a judge) and the K-Precision of the summarizer at guessing the question. However, an alternative summarizer implementation could guess *multiple* potential questions, which could all be compared against the user's question. Future work can investigate whether this approach can improve responses from H&S pipelines.

Automated checking for potentially adversarial knowledge. Our threat model assumes a trustworthy knowledge base. While this is realistic in many real-world use cases (e.g., Q&A based on FAQs), it may not be the case in all RAG applications. However, H&S makes it significantly easier to look for threats: it reduces the scope of the problem from inspecting *both* the knowledge base and the very large space of possible user inputs, down to inspecting only the knowledge base. For example, the system designer can scan their knowledge base for potential triggers, by inspecting each document via a (sequential) sliding window based of what the highlighter is allowed to highlight. Future work can investigate efficient techniques for such scanning.

Handling Yes/No questions. For some questions, a simple "Yes" or "No" can be a satisfactory answer. We observe that H&S can be easily augmented to support such questions, whilst ensuring the same level of security. For example, the highlighter can optionally pass a tag to the summarizer indicating: i) whether the question being asked is of yes-no type, and if so ii) what it thinks to be the answer (yes/no). Since both of these fields are of boolean type, they are unlikely to introduce any new security risk. The summarizer can then be instructed to augment its answer with this information. Future work can investigate whether this approach improves answers for this class of questions.

Handling multiple questions. In some cases, the user may ask more than one question in the same prompt. It may be possible to handle this via system design. For example, the highlighter can first split the questions that are present in the prompt, and then have the H&S system make separate calls to the summarizer. Future work can investigate whether the highlighter can accurately detect and separate multiple questions.

Does H&S reduce hallucinations? Intuitively, H&S should reduce hallucinations compared to standard RAG pipelines by sticking strictly to the documents retrieved from the knowledge base. However, it is still possible that either the highlighter or summarizer could suffer from hallucinations. For example, as discussed in section 6, when faced with an unanswerable question, the highlighter sometimes highlighted sections that are either irrelevant or misleading. Furthermore, the summarizer may suffer from hallucinations in its generative step, as usual. Future work can investigate whether H&S, or some variant thereof, can help to reduce hallucinations.

10 Conclusion

We propose Highlight&Summarize (H&S), a design pattern that enhances the generative step of a RAG pipeline to provide security by design against jailbreaking and model hijacking attacks. Our empirical evaluations demonstrate that, compared to current RAG pipelines, our approach actually *improves* the accuracy of responses. We also discuss several new research questions that arise from this design pattern, and encourage their study as exciting avenues for future research.

Acknowledgments

We are grateful to Sahar Abdelnabi and Santiago Zanella-Béguelin for helpful discussions. We thank Elisa Alessandrini for help with revising this manuscript.

References

[1] Sahar Abdelnabi, Aideen Fay, Ahmed Salem, Egor Zverev, Kai-Chieh Liao, Chi-Huang Liu, Chun-Chih Kuo, Jannis Weigend, Danyael Manlangit, Alex Apostolov, et al. 2025. LLMail-Inject: A Dataset from a Realistic Adaptive Prompt Injection Challenge. *arXiv preprint arXiv:2506.09956* (2025).

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[3] Vaibhav Adlakha, Parishad BehnamGhader, Xing Han Lu, Nicholas Meade, and Siva Reddy. 2024. Evaluating correctness and faithfulness of instruction-following models for question answering. *Transactions of the Association for Computational Linguistics* 12 (2024), 681–699.

[4] Gabriel Alon and Michael Kamfonas. 2023. Detecting Language Model Attacks with Perplexity. arXiv:2308.14132 [cs.CL] https://arxiv.org/abs/2308.14132

[5] Luca Beurer-Kellner, Beat Buesser, Ana-Maria Creţu, Edoardo Debenedetti, Daniel Dobos, Daniel Fabian, Marc Fischer, David Froelicher, Kathrin Grosse, Daniel Naeff, Ezinwanne Ozoani, Andrew Paverd, Florian Tramèr, and Václav Volhejn. 2025. Design Patterns for Securing LLM Agents against Prompt Injections. arXiv:2506.08837 [cs.LG] https://arxiv.org/abs/2506.08837

[6] James P Callan. 1994. Passage-level evidence in document retrieval. In *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*. Springer, 302–310.

[7] Cheng-Han Chiang and Hung-yi Lee. 2023. Can large language models be an alternative to human evaluations? *arXiv preprint arXiv:2305.01937* (2023).

[8] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, et al. 2024. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*.

[9] Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. 2025. Securing AI Agents with Information-Flow Control. arXiv:2505.23643 [cs.CR] https://arxiv.org/abs/2505.23643

[10] Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. 2025. Defeating Prompt Injections by Design. arXiv:2503.18813 [cs.CR] https://arxiv.org/abs/2503.18813

[11] Arpad E Elo. 1967. The proposed uscf rating system, its development, theory, and applications. *Chess life* 22, 8 (1967), 242–247.

[12] Zhengmian Hu, Gang Wu, Saayan Mitra, Ruiyi Zhang, Tong Sun, Heng Huang, and Viswanathan Swaminathan. 2024. Token-Level Adversarial Prompt Detection Based on Perplexity Measures and Contextual Information. arXiv:2311.11509 [cs.CL] https://arxiv.org/abs/2311.11509

[13] Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. 2023. Baseline Defenses for Adversarial Attacks Against Aligned Language Models. arXiv:2309.00614 [cs.LG] https://arxiv.org/abs/2309.00614

[14] Changyue Jiang, Xudong Pan, Geng Hong, Chenfu Bao, and Min Yang. 2024. RAG-Thief: Scalable Extraction of Private Data from Retrieval-Augmented Generation Applications with Agent-based Attacks. arXiv:2411.14110 [cs.CR] https://arxiv.org/abs/2411.14110

[15] Jing Jiang and Chengxiang Zhai. 2006. Extraction of coherent relevant passages using hidden markov models. *ACM Transactions on Information Systems (TOIS)* 24, 3 (2006), 295–319.

[16] Daniel Jurafsky and James H. Martin. 2025. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models* (3rd ed.). https://web.stanford.edu/~jurafsky/slp3/ Online manuscript released January 12, 2025.

[17] Aounon Kumar, Chirag Agarwal, Suraj Srinivas, Aaron Jiaxun Li, Soheil Feizi, and Himabindu Lakkaraju. 2025. Certifying LLM Safety against Adversarial Prompting. arXiv:2309.02705 [cs.CL] https://arxiv.org/abs/2309.02705

[18] Kenton Lee, Shimi Salant, Tom Kwiatkowski, Ankur Parikh, Dipanjan Das, and Jonathan Berant. 2016. Learning recurrent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436* (2016).

[19] Microsoft. 2025. Safety system messages. https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/system-message

[20] Joao Monteiro, Pierre-Andre Noel, Etienne Marcotte, Sai Rajeswar Mudumba, Valentina Zantedeschi, David Vazquez, Nicolas Chapados, Chris Pal, and Perouz Taslakian. 2024. RepliQA: A question-answering dataset for benchmarking llms on unseen reference content. *Advances in Neural Information Processing Systems* 37 (2024), 24242–24276.

[21] Norman Mu, Jonathan Lu, Michael Lavery, and David Wagner. 2025. A Closer Look at System Prompt Robustness. arXiv:2502.12197 [cs.CL] https://arxiv.org/abs/2502.12197

[22] John O'Connor. 1975. Retrieval of answer-sentences and answer-figures from papers by text searching. *Information Processing & Management* 11, 5-7 (1975),

155–164.
[23] John O'Connor. 1980. Answer-passage retrieval by text searching. *Journal of the American Society for Information Science* 31, 4 (1980), 227–239.
[24] Harrie Oosterhuis, Rolf Jagerman, Zhen Qin, Xuanhui Wang, and Michael Bendersky. 2024. Reliable confidence intervals for information retrieval evaluation using generative ai. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.
[25] OpenAI. [n. d.]. Moderation. https://platform.openai.com/docs/guides/moderation/overview
[26] Kate Pearce, Tiffany Zhan, Aneesh Komanduri, and Justin Zhan. 2021. A comparative study of transformer-based language models on extractive question answering. *arXiv preprint arXiv:2110.03142* (2021).
[27] Yuefeng Peng, Junda Wang, Hong Yu, and Amir Houmansadr. 2025. Data Extraction Attacks in Retrieval-Augmented Generation via Backdoors. arXiv:2411.01705 [cs.CR] https://arxiv.org/abs/2411.01705
[28] Matthew Pisano, Peter Ly, Abraham Sanders, Bingsheng Yao, Dakuo Wang, Tomek Strzalkowski, and Mei Si. 2024. Bergeron: Combating Adversarial Attacks through a Conscience-Based Alignment Framework. arXiv:2312.00029 [cs.CR] https://arxiv.org/abs/2312.00029
[29] Archiki Prasad, Trung Bui, Seunghyun Yoon, Hanieh Deilamsalehy, Franck Dernoncourt, and Mohit Bansal. 2023. MeetingQA: Extractive question-answering on meeting transcripts. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 15000–15025.
[30] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Jian Su, Kevin Duh, and Xavier Carreras (Eds.). Association for Computational Linguistics, Austin, Texas, 2383–2392. doi:10.18653/v1/D16-1264 arXiv:1606.05250 [cs.CL]
[31] Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. 2023. NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails. arXiv:2310.10501 [cs.CL] https://arxiv.org/abs/2310.10501
[32] Alexander Robey, Eric Wong, Hamed Hassani, and George J. Pappas. 2024. SmoothLLM: Defending Large Language Models Against Jailbreaking Attacks. arXiv:2310.03684 [cs.LG] https://arxiv.org/abs/2310.03684
[33] Hovav Shacham. 2007. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*. doi:10.1145/1315245.1315313
[34] Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. 2024. Replacing judges with juries: Evaluating LLM generations with a panel of diverse models. *arXiv preprint arXiv:2404.18796* (2024).
[35] Luqi Wang, Kaiwen Zheng, Liyin Qian, and Sheng Li. 2022. A survey of extractive question answering. In *2022 International Conference on High Performance Big Data and Intelligent Systems (HDIS)*. IEEE, 147–153.
[36] Baolei Zhang, Yuxi Chen, Minghong Fang, Zhuqing Liu, Lihai Nie, Tong Li, and Zheli Liu. 2025. Practical Poisoning Attacks against Retrieval-Augmented Generation. arXiv:2504.03957 [cs.CR] https://arxiv.org/abs/2504.03957
[37] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems* (2023).
[38] Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. 2024. PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models. arXiv:2402.07867 [cs.CR] https://arxiv.org/abs/2402.07867

A Additional results

Table 9: Pairwise battle results: Each cell shows (Wins, Ties) for the row model vs the column model. The last columns show total wins and ties.

RepliQA												
Opponent	RAG		DeBERTaV3 (SQuAD2)		Baseline		Structured		DeBERTaV3 (RepliQA)		Total	
	Wins	Ties	Wins	Ties	Wins	Ties	Wins	Ties	Wins	Ties	Wins	Ties
RAG	–	–	13123	2443	1076	3587	1926	3732	6753	3113	22878	12875
DeBERTaV3 (SQuAD2)	2389	2443	–	–	532	1959	629	2001	2469	4694	6019	11097
Structured	13292	3587	15464	1959	–	–	4510	11261	9089	7355	42355	24162
Baseline	12297	3732	15325	2001	2184	11261	–	–	8856	6186	38662	23180
DeBERTaV3 (RepliQA)	8089	3113	10792	4694	1511	7355	2913	6186	–	–	23305	21348
BioASQ												
Opponent	RAG		DeBERTaV3 (SQuAD2)		Baseline		Structured		DeBERTaV3 (RepliQA)		Total	
	Wins	Ties	Wins	Ties	Wins	Ties	Wins	Ties	Wins	Ties	Wins	Ties
RAG	–	–	3837	340	746	546	437	506	2924	457	7944	1849
DeBERTaV3 (SQuAD2)	542	340	–	–	290	510	140	394	921	1455	1893	2699
Baseline	3427	546	3919	510	–	–	955	2458	3465	723	11766	4237
Structured	3776	506	4185	394	1306	2458	–	–	3682	776	12949	4134
DeBERTaV3 (RepliQA)	1338	457	2343	1455	531	723	261	776	–	–	4473	3411